UNITED STATES PATENT APPLICATION

INDIRECTLY ADDRESSED VECTOR LOAD-OPERATE-STORE METHOD AND APPARATUS

INVENTOR:

James R. Kohn Citizenship: **USA** Residence: **Inver Grove Heights, MN.**

Post Office Address: **10165 Adam Avenue, Inver Grove Heights, MN. 55077**

Schwegman, Lundberg, Woessner, & Kluth, P.A.

1600 TCF Tower

121 South Eighth Street

Minneapolis, Minnesota 55402

Attorney Docket 01376.730us1

INDIRECTLY ADDRESSED VECTOR LOAD-OPERATE-STORE METHOD AND APPARATUS

Related Applications

This application is related to U.S. Patent Application No. _____, entitled "Multistream Processing System and Method", filed on even date herewith; to U.S. Patent Application No. _____, entitled "System and Method for Synchronizing Memory Transfers", Serial No. _____, filed on even date herewith; to U.S. Patent Application No. _____, entitled "Decoupled Store Address and Data in a Multiprocessor System", filed on even date herewith; to U.S. Patent Application No. _____, entitled "Decoupled Vector Architecture", filed on even date herewith; to U.S. Patent Application No. _____, entitled "Latency Tolerant Distributed Shared Memory Multiprocessor Computer", filed on even date herewith; to U.S. Patent Application No. _____, entitled "Relaxed Memory Consistency Model", filed on even date herewith; to U.S. Patent Application No. _____, entitled "Remote Translation Mechanism for a Multinode System", filed on even date herewith; and to U.S. Patent Application No. _____, entitled "Method and Apparatus for Local Synchronizations in a Vector Processor System", filed on even date herewith, each of which is incorporated herein by reference.

Field of the Invention

This invention relates to the field of vector computers, and more specifically to a method and apparatus to correctly computer a vector-load, vector-operate (such as a vector add), and vector-store sequence, particularly when elements of the vector may be redundantly presented as in the case of indirectly addressed vector operations from and to memory.

Background of the Invention

Indirectly addressed operands are frequently used in computer programs. For example, one typical situation provides a load instruction that specifies a register having an address of an operand in memory (rather than the address being partially or completely specified directly by the instruction), and another register that is the destination of the operand being fetched or loaded. A store instruction using indirect addressing would similarly specify a register that holds the address in memory of the destination, and another register that is the source of the operand being stored.

Vector computers provide a fast and compact way of programming for codes that are amenable to vectorizing to improve speed and programming efficiency.

What is needed is a fast, repeatable, and accurate way of performing various indirectly addressed operations in a vector computer.

Summary of the Invention

The present invention provides a method and apparatus to correctly compute a vector-gather, vector-operate (e.g., vector add), and vector-scatter sequence, particularly when elements of the vector may be redundantly presented, as with indirectly addressed vector operations. For an add operation, one vector register is loaded with the "add-in" values, and another vector register is loaded with address values of "add to" elements to be gathered from memory into a third vector register. If the vector of address values has a plurality of elements that point to the same memory address, the algorithm should add all the "add in" values from elements corresponding to the elements having the duplicated addresses. An indirectly addressed load performs the "gather" operation to load the "add to" values. A vector add operation then adds corresponding elements from the "add in" vector to the "add to" vector. An indirectly addressed store then performs the "scatter" operation to store the results.

Brief Description of the Drawings

FIG. 1A shows a block diagram of one embodiment of the present invention having a vector processing system 100.

FIG. 1B shows a block diagram of further aspects of vector processing system 100.

Fig. 1C shows a block diagram of an MSP 102 of some embodiments of the present invention.

Fig. 1D shows a block diagram of a node 106 of some embodiments of the present invention.

Fig. 1E shows a block diagram of a system 108 of some embodiments of the present invention.

Description of Preferred Embodiments

In the following detailed description of the preferred embodiments, reference is made to the accompanying drawings that form a part hereof, and in which are shown by way of illustration specific embodiments in which the invention may be practiced. It is understood that other embodiments may be utilized and structural changes may be made without departing from the scope of the present invention.

The leading digit(s) of reference numbers appearing in the Figures generally corresponds to the Figure number in which that component is first introduced, such that the same reference number is used throughout to refer to an identical component which appears in multiple Figures. The same reference number or label may refer to signals and connections, and the actual meaning will be clear from its use in the context of the description.

In some embodiments, there is a software application invention that requires that particular sequence of operations to maintain order. The algorithm itself has been vectorized. There is a requirement that within a vector of more than one element, since there may be collisions in the memory system where its referencing the same memory location multiple times in the vector. There needs to be a guarantee that updates to that memory location are done in order. In some embodiments, each memory location is an 8-byte memory location. In some embodiments, there is a vector instruction that can operate on multiple 8 byte

quantities in one instruction, by using one vector register to hold addresses of the elements to be loaded into another vector register, that may not be contiguous in memory. In fact in this particular case they are often not contiguous.

Memory location can be referenced with a single instruction. But there may be multiple occurrences of any given memory address in that single instruction and now we're trying to do like a add of a value to that memory location. And when an addition operation occurs multiple times, there is the possibility of losing one of the adds or getting the adds out-of-order. This has been a known vectorization problem.

There are generally three instructions of interest. There is a load operation which loads the existing memory contents or a number of elements greater than one, into a vector register using indirect addressing. Then there's an add operation that wants to add a set of values to those elements that are loaded, such as V1 is assigned V2 plus V1. Then we want to store the result back out into the same memory location. And if the memory locations are all disjoint, this can occur at full speed in the vector hardware of Figure 1D described below. The problem occurs, for which this special algorithm is needed, is when there are overlapping or multiple occurrences of the same memory location in the vector register used for addressing. The original values are loaded into v1. Now we add v2 to v1. In conventional methods, the first element that has multiple instances of the address is correct, but the additions after that are or can be incorrect because they lose the previous additions. So when we store the final result of the add back out to memory, we get an incorrect answer in memory. Thus, we need a method to recognize where the conflicting memory locations are, and we have such an algorithm for older systems, and part of the application is probably going to have to describe that old algorithm. And then for the X1 that old algorithm did not work very well, the present invention provides a new way of detecting those collisions.

In one conventional algorithm, after you did that load from memory, you would use the same memory location to store back a known pattern and then you would load back that pattern and do a comparison against the original pattern and if

they matched then there were no collisions. But if they didn't match that means that one or more locations, or that a location had more than one store into it.

The other vector register specifies an index to those locations. And it's those indexes that may be repeated. That index is used both for the load as well as the store back later.

In the old way what you'd do is you'd have a pattern of say 1,2,3,4,5,6,7 in the elements and if you didn't get back, if you got 1,2,2 or 1,6,6. You would see where there was a collision and which elements were colliding. Then you unwrap the vector and do it as individual instructions. Effectively that's the conventional algorithm. The intent is to have a fast way of detecting that we do have a collision. The new algorithm, instead of using the original array that we loaded, storing this 1,2,3,4,5 etc., creates a temporary scratch array and uses that instead.

In fact, one can delay the load of the elements to be added, since the calculations to determine duplicates only needs the scratch area and the addressing vector register. The algorithm selects a certain number of bits out of the index vector elements, like say 12 bits, it doesn't' really matter how many bits, and use that reduced index of the index into the temporary. Occasionally you get some false positives. The new algorithm addresses how to deal with the false positives. And does it in such a way that performance is improved on the X1 with this new technique.

The new algorithm goes on, instead of doing an add like the old algorithm did, it does an add back into the add-in values having duplicated indexes to compress that vector.

Figure 1A shows a block diagram of one embodiment of the present invention having a vector-processing system 100. Figure 1B shows a block diagram of further aspects of vector processing system 100.

In some embodiments, as shown in Figures 1A and 1B, a first vector register 110 having E elements is loaded with the "add-in" values A0, A1, ... A(E-1) into element addresses 0, 1, ... (E-1) of register 110 (i.e., each element of the first vector

register 110 is a different value to be added to a corresponding element fetched from memory), and a second vector register 112 is loaded with address values @0, @1, ... @(E-1)(i.e., each element of the second vector register 112 is a different signed offset value to be added to a base address pointer to obtain the virtual address of the corresponding element fetched from memory), of "add to" elements to be gathered from memory 150 (e.g., from a table).

Occasionally, a plurality of such addresses will be equal, thus specifying to fetch the same "add to" element to a plurality of locations in the add-to vector register 110. For example, elements 2, 15, and 47 (these are the element addresses of elements in the vector) of the second register 112 might all have the same offset, say 60033, and the base register could have the pointer to, say address 500000. Thus, the addresses of elements 2, 15, and 47 would each point to memory address 560033. The elements 2, 15, and 47 of the "add to" vector 110 would all be loaded with the value from memory address 5033.

In some embodiments, the desired algorithm would want the same behavior and the same result whether the gather-add-scatter operations were performed one element at a time, 16 elements at a time, 64 elements at a time, or any other number of elements at a time, and regardless of the alignment of the gathered elements relative to the start of any vector operation. Thus, in this example, the value starting in memory address 560033 would be loaded (in a vector "gather" operation), be added to the "add in" values from elements 2, 15, and 47 of the first vector register 110, and this combined result would be stored back to memory location 560033 (in a "scatter" operation). In some embodiments, this provides the same result as if the value starting in memory address 560033 would be loaded (in a serial "gather" operation), be added to the "add in" value from element 2 of the first vector register 110, and stored back to memory location 560033, then this value from memory address 560033 would be again loaded, be added to the "add in" value from element 15 of the first vector register 110, and stored back to memory location 560033, and then this value from memory address 560033 would be again loaded, be added to the

"add in" value from element 47 of the first vector register 110, and stored back to memory location 560033.

Since the identities of the elements in the second vector register 112 having the same addresses are unknown, the present invention provides a way to determine those elements. In some embodiments, a first sequence of identification values is stored to a series of addressed locations within a constrained area of memory 161. The address of each location used to store the sequence of values in the constrained area 161 is based at least in part on a corresponding one of the addressing values. For example, the constrained area could have 2^N locations (e.g., in some embodiments, $2^N = 2^{12} = 4096$ locations), and N bits (e.g., N=12 bits) of the address value are used as an offset into the constrained area. Continuing with the above example, the address offset 60033 could have any 12 bits extracted, Assume, for example, the low 12 bits are used, which would extract "033" from the 60033 value, assuming hexadecimal number notation. If the constrained area 161 had a base address of 7000, then the location 7033 would be the destination of the identification values for elements 2, 15, and 47, and since they are written in element order, location 7033 would end up with the value stored for element 47.

The method then reads back 116 from the sequence of addressed locations values resulting from the storing of the first sequence to obtain a second sequence of values, comparing 118 the first sequence of values to the second sequence of values to generate a bit vector representing compares and miscompares, compressing 120 the second vector of operand values using the bit vector, using the first vector of addressing values as masked by the bit vector. I.e., for an add operation, where the redundantly addressed locations point to a single memory value B(m) (i.e., at location 560033), each of the corresponding A elements 2, 15, and 47 are added to that Bm value and the result is stored to location 56033). The method further includes loading 124 a third vector register with elements from memory, performing 126 an arithmetic-logical operation using values from the third vector register and the compressed second vector of operand values to generate a result vector, and

using the first vector of addressing values as masked by the bit vector, storing 128 the result vector to memory.

One exemplary program that codes one embodiment of the invention is listed below.

Figure 1C shows a block diagram of a multistreaming processor (MSP) 102 that is usable by the above method, for some embodiments of the present invention. MSP 102 includes a plurality of P chips or P circuits 100 (each representing one single-streaming processor having a plurality of vector pipelines and a scalar pipeline), each P chip/circuit 100 connected to a plurality of E chips or E circuits 101 (each representing an external cache, synchronization, and memory-interface function). In some embodiments, every P chip/circuit 100 is connected to every E chip/circuit 101. In some embodiments, four P Chips 100 and four E Chips 101 form one MSP 102. Although the P Chip 100 and the E Chips 101 are sometimes described herein as "chips" as representing one embodiment, in other embodiments, they are implemented with a plurality of chips each, or with a single chip containing a plurality of P circuits 100 and/or E circuits 101.

In some embodiments, each scalar processing unit 12 delivers a peak of 0.4 GFLOPS and 0.8 GIPS at the target frequency of 400 MHz. Each processor 100 contains two vector pipes, running at 800 MHz, providing 3.2 GFLOPS for 64-bit operations and 6.4 GFLOPS for 32-bit operations. The MSP 102 thus provides a total of 3.2 GIPS and 12.8/25.6 GFLOPS. Each processor 100 contains a small Dcache used for scalar references only. A two-MB Ecache 24 is shared by all the processors 100 in MSP 102 and used for both scalar and vector data. In one embodiment, each processor 100 and e-circuit 101 of cache 24 are packaged as separate chips (termed the "P" chip and "E" chips, respectively).

In some embodiments, signaling between processor 100 and cache 24 runs at 400 Mb/s on processor-to-cache connection 32. Each processor-to-cache connection 32 shown in Figure 1C uses an incoming 64-bit path for load data and an outgoing 64-bit path for requests and store data. Loads, in some embodiments, can achieve a

maximum transfer rate of fifty-one GB/s from cache 24. Stores, in some embodiments, can achieve up to forty-one GB/s for stride-one and twenty-five GB/s for non-unit stride stores.

In some embodiments, global memory 26 is distributed to each MSP 102 as local memory 105. Each E Chip 101 has four ports 34 to M chip 104 (and through M chip 104 to local memory 105 and to network 107). In some embodiments, ports 34 are sixteen data bits in each direction. MSP 102 has a total of 25.6 GB/s load bandwidth and 12.8-20.5 GB/s store bandwidth (depending upon stride) to local memory.

Figure 1D shows a block diagram of a node 106 of some embodiments of the present invention. In some embodiments, a node 106 is packaged on a single printed-circuit board. Node 106 includes a plurality of MSPs 102 each connected to a plurality of M chips 104, each M-chip 104 controlling one or more sections of memory 105. In some embodiments, each M chip 104 is connected to memory 105 using a plurality of channels (e.g., eight), each channel having a plurality of direct RAMBUS DRAM chips (e.g., four). In some embodiments, each node also includes a plurality of I/O channels 103 used to connect to a local-area network (e.g., one or more gigabit ethernet connections) and/or storage (e.g., disk storage or a storage-area network). Each node 106 also includes one or more network connections that interconnect the memories of a plurality of nodes, in some embodiments.

In some embodiments, each node 106 includes four MSPs 102 and sixteen M chips 104. M chips 104 contain memory controllers, network interfaces and cache coherence directories with their associated protocol engines. In one such embodiment, memory 26 is distributed round-robin by 32-byte cache lines across the sixteen M chips 104 at each node 106. Thus, the M chip for a particular address is selected by bits 8..5 of the physical address.

Each E Chip 101 is responsible for one fourth of the physical address space, determined by bits 5 and 6 of the physical address. A reference to a particular line of memory is sent to the associated E Chip 101 where the Ecache is consulted, and either the line is found in the Ecache or the request is sent on to an M chip. Bits 7

and 8 of the physical address select one of four M chips connected to each E Chip 101.

Each M chip 104 resides in one of sixteen independent slices of the machine, and the interconnection network 107 provides connectivity only between corresponding M chips on different nodes (thus there are sixteen parallel, independent networks). All activity (cache, memory, network) relating to a line of memory stays within the corresponding system slice.

Each M chip 104 contains two network ports 44, each 1.6 GB/s peak per direction. This provides a total peak network bandwidth of 51.2 GB/s in and 51.2 GB/s out. Single transfers to/from any single remote destination will use only half this bandwidth, as only one of two ports 44 per M chip 104 will be used. Also, contention from the other processors 100 on node 106 must be considered. Lastly, all inter-node data is packetized, resulting in a smaller ratio of sustained to peak than in the local memory subsystem. Protocol overheads vary from 33% (one way, stride-1 reads) to 83% (symmetric, non-unit-stride reads or writes).

Each node 106 also contains two I/O controller chips 103 ("I" chips) that provide connectivity between the outside world and network 107 and memory 26. In some embodiments, each "I" chip 103 provides two XIO (a.k.a. Crosstalk) I/O channels 49, with a peak speed bandwidth of 1.2 GB/s full duplex each. The I chips are connected to each other and to the sixteen M chips 104 with enough bandwidth to match the four XIO channels.

This partitioning provides low latency and high bandwidth to local memory 105. With a local memory size of up to sixteen GB (sixty-four GB, once 1 Gbit chips become available), most single-processor and autotasked codes should run locally, and most references in distributed-memory codes will be satisfied locally as well. Latency to remote memory will depend upon the distance to the remote node, and the level of contention in network 107.

In some embodiments, a limited operating system executes on each node, with a Unicos/mk-like layer across nodes 106. The limited OS will provide basic kernel services and management of two direct-attached I/O devices (a disk array and

network interface). All other I/O connectivity is provided by a separate host system. In one such embodiment, the host system also provides the user environment (shell, cross compilers, utility programs, etc.), and can be used to run scalar compute applications.

Figure 1E shows a block diagram of a system 108 of some embodiments of the present invention. System 108 includes a plurality of nodes 106 each connected to a common network 107. In some embodiments, network 107 is also connected to one or more other networks 109.

One aspect of the invention provides a computerized method that includes providing a first vector 110 of addressing values, providing a second vector 112 of operand values, storing 114 a first sequence of values to a sequence of addressed locations within a constrained area of memory, wherein each location's address is based at least in part on a corresponding one of the addressing values, reading back 116 from the sequence of addressed locations values resulting from the storing of the first sequence to obtain a second sequence of values, comparing 118 the first sequence of values to the second sequence of values to generate a bit vector representing compares and miscompares, compressing 120 the second vector of operand values using the bit vector, using the first vector of addressing values as masked by the bit vector, loading 124 a third vector register with elements from memory, performing 126 an arithmetic-logical operation using values from the third vector register and the compressed second vector of operand values to generate a result vector, and using the first vector of addressing values as masked by the bit vector, storing 128 the result vector to memory.

In some embodiments, addresses of the elements in memory are calculated by adding each respective addressing value to a base address of an object in memory.

In some embodiments, the arithmetic-logical operation is an addition operation that produces at least one element of the result vector as a summation of an element of the loaded third vector register and a plurality of respective elements of the original second vector of operand values corresponding to elements of the first vector of addressing values that had identical values.

In some embodiments, address values for the sequence of addressed locations within the constrained area of memory are each calculated using a truncated portion of each respective addressing value of the first vector of addressing values. In some embodiments, data values of the first sequence of values are each formed by concatenating a portion of each respective addressing value of the first vector of addressing values to a respective one of a sequence of numbers.

In some embodiments, the constrained area of memory includes 2^N locations, wherein address values for the sequence of addressed locations within the constrained area of memory are each calculated by adding a base address to an N-bit portion of each respective addressing value of the first vector of addressing values, and wherein data values of the first sequence of values are each formed by concatenating a portion of each respective addressing value of the first vector of addressing values to a respective one of a consecutive sequence of integer numbers.

In some embodiments, for the loading of the third vector register with elements from memory, elements are loaded from locations specified by addressing values corresponding to bits of the bit vector that indicated a compare and no elements are loaded from locations specified by addressing values corresponding to bits of the bit vector that indicated a miscompare.

In some embodiments, the operations recited therein are executed in the order recited therein.

Some embodiments, further include performing 124 a first synchronization operation that ensures that the comparing the first sequence of values to the second sequence of values to generate the bit vector representing compares and miscompares effectively completes before the loading of the third vector register with elements from memory, and performing 130 a second synchronization operation that ensures that the storing the result vector to memory completes before subsequent passes through a loop.

Another aspect of the invention provides a computer-readable medium having instructions stored thereon for causing a suitably programmed information-processing system to execute a method that includes providing 110 a first vector of

addressing values, providing 112 a second vector of operand values, storing 114 a first sequence of values to a sequence of addressed locations within a constrained area of memory, wherein each location's address is based at least in part on a corresponding one of the addressing values, reading back 116 from the sequence of addressed locations values resulting from the storing of the first sequence to obtain a second sequence of values, comparing 118 the first sequence of values to the second sequence of values to generate a bit vector representing compares and miscompares, compressing 120 the second vector of operand values using the bit vector, using the first vector of addressing values as masked by the bit vector, loading 124 a third vector register with elements from memory, performing 126 an arithmetic-logical operation using values from the third vector register and the compressed second vector of operand values to generate a result vector, and using the first vector of addressing values as masked by the bit vector, storing 128 the result vector to memory.

Yet another aspect of the invention provides a computerized method that includes loading 210 a first vector register with addressing values, loading 212 a second vector register with operand values, storing 214 a first sequence of values to a sequence of addressed locations within a constrained area of memory, wherein each one of these location's addresses in the constrained area of memory is based at least in part on a subset of bits of a corresponding one of the addressing values, reading back 216 from the sequence of addressed locations values resulting from the storing of the first sequence to obtain a second sequence of values, comparing 218 the first sequence of values to the second sequence of values, selectively combining 220, with an arithmetic-logical operation, certain elements of the second vector of operand values based on results of the comparing, using at least some of the first vector register of addressing values, loading 224 a third vector register with elements from memory, performing 226 the arithmetic-logical operation using values from the third vector register and the combined second vector of operand values to generate a result vector, and using the at least some of the first vector register of addressing values, storing 228 the result vector to memory.

In some embodiments, addresses of the elements from memory are calculated by adding each respective addressing value to a base address.

In some embodiments, addresses of the elements from memory are calculated by performing a signed-addition operation of each respective addressing value to a base address of an object in memory.

In some embodiments, the arithmetic-logical operation is an addition operation that produces at least one element of the result vector as a summation of an element of the loaded third vector register and a plurality of respective elements of the original second vector of operand values corresponding to elements of the first vector register of addressing values having identical values.

In some embodiments, address values for the sequence of addressed locations within the constrained area of memory are each calculated using a truncated portion of each respective addressing value of the first vector register of addressing values.

In some embodiments, data values of the first sequence of values are each formed by concatenating a portion of each respective addressing value of the first vector register of addressing values to a respective one of a sequence of numbers.

In some embodiments, the constrained area contains 2^N consecutive addresses, wherein address values for the sequence of addressed locations within the constrained area of memory are each calculated using an N-bit value derived from each respective addressing value of the first vector register of addressing values, and wherein data values of the first sequence of values are each formed by concatenating a portion of each respective addressing value of the first vector register of addressing values to a respective one of a consecutive sequence of integer numbers.

In some embodiments, for the loading of the third vector register with elements from memory, elements are loaded from locations specified by addressing values corresponding to indications that indicated compares and no elements are loaded from locations specified by addressing values corresponding to indications that indicated miscompares.

Another aspect of the invention provides a computer-readable medium having instructions stored thereon for causing a suitably programmed information-

processing system to execute one or more of the various embodiments of the above method.

In some embodiments, the constrained area contains 2^N consecutive addresses, address values for the sequence of addressed locations within the constrained area of memory are each calculated using an N-bit value derived from each respective addressing value of the first vector register of addressing values, data values of the first sequence of values are each formed by combining at least a portion of each respective addressing value of the first vector register of addressing values to a respective one of a consecutive sequence of integer numbers, for the loading of the third vector register with elements from memory, elements are loaded from locations specified by addressing values corresponding to indications that indicated compares and no elements are loaded from locations specified by addressing values corresponding to indications that indicated miscompares, addresses of the elements from memory are calculated by adding each respective addressing value to a base address, the arithmetic-logical operation is a floating-point addition operation that produces at least one element of the result vector as an ordered-operation floating point summation of an element of the loaded third vector register and a plurality of respective elements of the original second vector of operand values corresponding to elements of the first vector register of addressing values having identical values, and for the storing of the result vector of elements to memory, elements are stored to locations specified by addressing values corresponding to indications that indicated compares and no elements are stored to locations specified by addressing values corresponding to indications that indicated miscompares.

Another aspect of the invention provides a system that includes a first vector processor having a first vector register having addressing values, a second vector register having operand values, a third vector register, a bit vector register, circuitry that selectively stores a first sequence of values to a sequence of addressed locations within a constrained area of memory, wherein each location's address is based at least in part on a corresponding one of the addressing values, circuitry that selectively loads, from the sequence of addressed locations, values resulting from the

stores of the first sequence to obtain a second sequence of values, circuitry that selectively compares the first sequence of values to the second sequence of values to generate bit values into the bit vector register representing compares and miscompares, circuitry that selectively compresses the second vector of operand values using the values in the bit vector register, circuitry that selectively loads the third vector register with elements from memory addresses generated from the first vector register of addressing values as masked by the bit vector register, circuitry that selectively performs an arithmetic-logical operation on corresponding values from the third vector register and the compressed second vector of operand values to generate values of a result vector, and, circuitry that selectively stores the result vector to memory.

Some embodiments of this system further include circuitry to calculate addresses of the elements in memory by adding each respective addressing value to a base address value.

In some embodiments of this system, the arithmetic-logical operation is an addition operation that produces at least one element of the result vector as a summation of an element of the loaded third vector register and a plurality of respective elements of the original second vector of operand values corresponding to elements of the first vector register of addressing values that had identical values.

Some embodiments of this system further include circuitry to calculate address values for the sequence of addressed locations within the constrained area of memory using a truncated portion of each respective addressing value of the first vector register of addressing values.

Some embodiments of this system further include circuitry to generate data values of the first sequence of values by joining a portion of each respective addressing value of the first vector register of addressing values to a respective one of a sequence of numbers.

Some embodiments of this system further include circuitry to generate address values of the sequence of addressed locations within the constrained area of memory by adding a base address to an N-bit portion of each respective addressing

value of the first vector register of addressing values, and circuitry to generate data values of the first sequence of values by combining a portion of each respective addressing value of the first vector register of addressing values with a respective one of a consecutive sequence of integer numbers.

In some embodiments, the circuitry that selectively loads the third vector register with elements from memory only loads element from locations specified by addressing values corresponding to bits of the bit vector that indicated a compare.

Some embodiments further include synchronization circuitry that ensures that the comparing the first sequence of values to the second sequence of values to generate the bit vector representing compares and miscompares effectively completes before the loading of the third vector register with elements from memory, and that ensures that the storing the result vector to memory completes before subsequent passes through a loop.

Some embodiments further include a second vector processor having: a first vector register having addressing values, a second vector register having operand values, a third vector register, a bit vector register, circuitry that selectively stores a first sequence of values to a sequence of addressed locations within a constrained area of memory, wherein each location's address is based at least in part on a corresponding one of the addressing values, circuitry that selectively loads, from the sequence of addressed locations, values resulting from the stores of the first sequence to obtain a second sequence of values, circuitry that selectively compares the first sequence of values to the second sequence of values to generate bit values into the bit vector register representing compares and miscompares, circuitry that selectively compresses the second vector of operand values using the values in the bit vector register, circuitry that selectively loads the third vector register with elements from memory addresses generated from the first vector register of addressing values as masked by the bit vector register, circuitry that selectively performs an arithmetic-logical operation on corresponding values from the third vector register and the compressed second vector of operand values to generate values of a result vector, and, circuitry that selectively stores the result vector to memory. This system also

includes synchronization circuitry that ensures that the comparing the first sequence of values to the second sequence of values to generate the bit vector representing compares and miscompares effectively completes in both the first and second vector processors before the loading of the third vector register with elements from memory in either processor, and that ensures that the storing the result vector to memory completes before subsequent passes through a loop.

Another aspect of the invention provides a system that includes a first vector register, a second vector register, a third vector register, a bit vector register, means for loading the first vector register with addressing values, means as described herein for loading the second vector register with operand values, means for storing a first sequence of values to a sequence of addressed locations within a constrained area of memory, wherein each one of these location's addresses in the constrained area of memory is based at least in part on a subset of bits of a corresponding one of the addressing values, means for loading from the sequence of addressed locations values resulting from the storing of the first sequence to obtain a second sequence of values, means for comparing the first sequence of values to the second sequence of values, means for selectively combining, with an arithmetic-logical operation, certain elements of the second vector of operand values based on results of the comparing, means for loading a third vector register with elements from memory address locations generated using at least some of the first vector register of addressing values, means for performing the arithmetic-logical operation using values from the third vector register and the combined second vector of operand values to generate a result vector, and means for storing the result vector to memory.

Another aspect of the invention provides a system including a first vector register that can be loaded with addressing values, a second vector register that can be loaded with operand values, a third vector register that can be loaded with operand values from memory locations indirectly addressed using the addressing values from the first vector register, a circuit that determines element addresses of the first vector register that have a value that duplicates a value in another element address, a circuit that selectively adds certain elements of the second vector of

operand values based on the element addresses the duplicated values, a circuit that uses indirect addressing to selectively load the third vector register with elements from memory, a circuit that selectively adds values from the third vector register and the second vector of operand values to generate a result vector, and a circuit that selectively stores the result vector to memory using indirect addressing.

Some embodiments of this system further include an adder that generates addresses of the elements from memory by adding each respective addressing value to a base address.

Some embodiments of this system further include an adder that generates addresses of the elements from memory by a signed-addition operation of each respective addressing value to a base address of an object in memory.

In some embodiments, the circuit that selectively adds certain elements performs one or more addition operations using those values from a plurality of respective elements of the original second vector of operand values corresponding to elements of the first vector register of addressing values having identical values.

Multistreaming Aspects of Indirect Addressed Vector Add

Another aspect of the invention provides a computerized method that includes loading a first vector register with addressing values, loading a second vector register with operand values, determining which, if any, element addresses of the first vector register have a value that duplicates a value in another element address, selectively adding certain elements of the second vector of operand values based on the element addresses the duplicated values, loading, using indirect addressing from the first vector register, elements from memory into a third vector register, adding values from the third vector register and the second vector of operand values to generate a result vector, and storing the result vector to memory using indirect addressing.

In some embodiments, the set of operations (a), (b), (c), and (d) is performed substantially in parallel in the plurality of processors, and the set of operations (e), (f), and (g) is performed serially, one processor at a time.

Some embodiments further include executing an ordered Msync operation before the set of operations (e), (f), and (g), and executing an end ordered Msync operation after the set of operations (e), (f), and (g).

In some embodiments, the set of operations (a), (b), (c), and (d) is performed substantially in parallel in the plurality of processors.

Some embodiments of the method further include:

executing a first barrier synchronization operation before the set of operations (e), (f), and (g) in all of the plurality of processors,

executing a second barrier synchronization operation before the set of operations (e), (f), and (g) in the second processor,

executing the set of operations (e), (f), and (g) in the first processor and then executing a second barrier synchronization operation in the first processor to satisfy the second barrier synchronization in the second processor, and executing a third barrier synchronization in the first processor, and

executing the set of operations (e), (f), and (g) in the second processor and then executing a third barrier synchronization operation in the second processor to satisfy the third barrier synchronization in the first processor.

In some embodiments, the set of operations (a), (b), (c), and (d) is performed substantially in parallel in the plurality of processors.

In some embodiments, the determining of duplicates includes:

generating each respective address value for a sequence of addressed locations within a constrained area of memory containing 2^N consecutive addresses using an N-bit value derived from each respective addressing value of the first vector register,

generating each respective data value of a first sequence of values by combining at least a portion of each respective addressing value of the first vector register to a respective one of a sequence of integer numbers,

storing the first sequence of values to the constrained memory area

using the generated sequence of respective address values,

loading a second first sequence of values from the constrained memory area using the generated sequence of respective address values, and

comparing the first sequence of values to the second sequence of values, and

wherein the loading of the third vector register includes loading elements from locations specified by addressing values corresponding to indications of positive compares from the comparing,

wherein addresses of the elements from memory are calculated by adding each respective addressing value to a base address,

wherein the adding includes a floating-point addition operation that produces at least one element of the result vector as an ordered-operation floating point summation of an element of the loaded third vector register and a plurality of respective elements of the original second vector of operand values corresponding to elements of the first vector of addressing values having identical values, and

wherein for the storing of the result vector of elements to memory, elements are stored to locations specified by addressing values corresponding to indications of positive compares.

Another aspect of the invention provides a computerized method that includes:

(a) within a first vector processor:

loading a first vector register in the first vector processor with addressing values,

loading a second vector register in the first vector processor with operand values,

determining which, if any, element addresses of the first vector register in the first vector processor have a value that duplicates a value in another element address,

selectively adding certain elements of the second vector of operand values in the first vector processor based on the element addresses the duplicated values,

(b) within a second vector processor:

loading a first vector register in the second vector processor with addressing values,

loading a second vector register in the second vector processor with operand values,

determining which, if any, element addresses of the first vector register in the second vector processor have a value that duplicates a value in another element address,

selectively operating on certain elements of the second vector of operand values in the second vector processor based on the element addresses the duplicated values,

(c) performing a synchronization operation that ensures that prior store operations effectively complete in at least the second vector processor before the following (d) operations,

(d) within the first vector processor:

loading, using indirect addressing from the first vector register, elements from memory into a third vector register in the first vector processor,

operating on values from the third vector register and the second vector of operand values in the first vector processor to generate a first result vector, and

storing the first result vector to memory using indirect addressing.

(e) performing a synchronization operation that ensures that the storing of the first result vector effectively completes before the following (f) operations, and

(f) within the second vector processor:

loading, using indirect addressing from the first vector register, elements from memory into a third vector register in the second vector

processor,

operating on values from the third vector register and the second vector of operand values in the second vector processor to generate a second result vector, and

storing the second result vector to memory using indirect addressing.

In some embodiments, each of the "operating on" functions includes adding.

In some embodiments, the adding includes a floating-point addition operation that produces at least one element of the result vector as an ordered-operation floating point summation of an element of the loaded third vector register and a plurality of respective elements of the original second vector of operand values corresponding to elements of the first vector of addressing values having identical values.

In some embodiments, the determining of duplicates includes generating each respective address value for a sequence of addressed locations within a constrained area of memory containing 2^N consecutive addresses using an N-bit value derived from each respective addressing value of the first vector register, generating each respective data value of a first sequence of values by combining at least a portion of each respective addressing value of the first vector register to a respective one of a sequence of integer numbers, storing the first sequence of values to the constrained memory area using the generated sequence of respective address values, loading a second first sequence of values from the constrained memory area using the generated sequence of respective address values, and comparing the first sequence of values to the second sequence of values.

In some embodiments, the loading of the third vector register of each processor includes loading elements from locations specified by addressing values corresponding to indications of positive compares from the comparing operation.

In some embodiments, indirect addresses of the elements from memory are calculated by adding each respective addressing value to a base address.

One aspect of the invention provides a system that includes a first vector register having addressing values, a second vector register having operand values, circuitry programmed to determine which, if any, element addresses of the first vector register have a value that duplicates a value in another element address, circuitry programmed to selectively add certain elements of the second vector of operand values based on the element addresses the duplicated values, circuitry programmed to load, using indirect addressing from the first vector register, elements from memory into a third vector register, circuitry programmed to add values from the third vector register and the second vector of operand values to generate a result vector, and circuitry programmed to store the result vector to memory using indirect addressing.

In some embodiments, the circuitry programmed to determine duplicates further includes circuitry programmed to generate each respective address value for a sequence of addressed locations within a constrained area of memory containing 2^N consecutive addresses using an N-bit value derived from each respective addressing value of the first vector register, circuitry programmed to generate each respective data value of a first sequence of values by combining at least a portion of each respective addressing value of the first vector register to a respective one of a sequence of integer numbers, circuitry programmed to store the first sequence of values to the constrained memory area using the generated sequence of respective address values, circuitry programmed to load a second sequence of values from the constrained memory area using the generated sequence of respective address values, and circuitry programmed to compare the first sequence of values to the second sequence of values; and the circuitry programmed to load the third vector register loads elements from locations specified by addressing values corresponding to indications of positive compares; addresses of the elements from memory are calculated by adding each respective addressing value to a base address; and the circuitry programmed to add includes a floating-point adder that produces at least one element of the result vector as an ordered-operation floating point summation of an element of the loaded third vector register and a plurality of respective elements

of the original second vector of operand values corresponding to elements of the first vector of addressing values having identical values.

Some embodiments further include circuitry programmed to perform the set of operations (a), (b), (c), and (d) substantially in parallel in the plurality of processors, and circuitry programmed to perform the set of operations (e), (f), and (g) serially, one processor at a time.

Some embodiments further include circuitry programmed to execute an ordered Msync operation before the set of operations (e), (f), and (g); and circuitry programmed to execute an end ordered Msync operation after the set of operations (e), (f), and (g). Some such embodiments further include circuitry programmed to perform the set of operations (a), (b), (c), and (d) substantially in parallel in the plurality of processors.

Some embodiments further include circuitry programmed to execute a first barrier synchronization operation before the set of operations (e), (f), and (g) in all of the plurality of processors, circuitry programmed to execute a second barrier synchronization operation before the set of operations (e), (f), and (g) in the second processor, circuitry programmed to execute the set of operations (e), (f), and (g) in the first processor and then executing a second barrier synchronization operation in the first processor to satisfy the second barrier synchronization in the second processor, and executing a third barrier synchronization in the first processor, and circuitry programmed to execute the set of operations (e), (f), and (g) in the second processor and then executing a third barrier synchronization operation in the second processor to satisfy the third barrier synchronization in the first processor. Some such embodiments further include circuitry programmed to perform the set of operations (a), (b), (c), and (d) substantially in parallel in the plurality of processors.

Another aspect of the invention provides a system that includes

(a) a first vector processor including means as described herein for loading a first vector register in the first vector processor with addressing values, means for loading a second vector register in the first vector processor with operand values, means for determining which, if any, element addresses of the first vector register in the first

vector processor have a value that duplicates a value in another element address, and means for selectively adding certain elements of the second vector of operand values in the first vector processor based on the element addresses the duplicated values; and

(b) a second vector processor including means for loading a first vector register in the second vector processor with addressing values, means for loading a second vector register in the second vector processor with operand values, means for determining which, if any, element addresses of the first vector register in the second vector processor have a value that duplicates a value in another element address, means for selectively operating on certain elements of the second vector of operand values in the second vector processor based on the element addresses the duplicated values,

(c) means for performing a synchronization operation that ensures that prior store operations effectively complete in at least the second vector processors before the operations of the following (d) means,

(d) within the first vector processor: means for loading, using indirect addressing from the first vector register, elements from memory into a third vector register in the first vector processor, means for operating on values from the third vector register and the second vector of operand values in the first vector processor to generate a first result vector, and means for storing the first result vector to memory using indirect addressing;

(e) performing a synchronization operation that ensures that the storing of the first result vector effectively completes before the operations of the following (f) means, and

(f) within the second vector processor:

means for loading, using indirect addressing from the first vector register, elements from memory into a third vector register in the second vector processor,

means for operating on values from the third vector register and the second vector of operand values in the second vector processor to generate

a second result vector, and

means for storing the second result vector to memory using indirect addressing.

In some embodiments, each of the means for operating on functions includes an adder.

In some embodiments, wherein the adder includes a floating-point adder that produces at least one element of the result vector as an ordered-operation floating point summation of an element of the loaded third vector register and a plurality of respective elements of the original second vector of operand values corresponding to elements of the first vector of addressing values having identical values.

In some embodiments, wherein the means for determining of duplicates includes: means as described herein for generating each respective address value for a sequence of addressed locations within a constrained area of memory containing 2^N consecutive addresses using an N-bit value derived from each respective addressing value of the first vector register, means for generating each respective data value of a first sequence of values by combining at least a portion of each respective addressing value of the first vector register to a respective one of a sequence of integer numbers, means for storing the first sequence of values to the constrained memory area using the generated sequence of respective address values, means for loading a second first sequence of values from the constrained memory area using the generated sequence of respective address values, and means for comparing the first sequence of values to the second sequence of values.

In some embodiments, the means for loading of the third vector register of each processor includes means for loading elements from locations specified by addressing values corresponding to indications of positive compares from the comparing operation.

In some embodiments, indirect addresses of the elements from memory are calculated by adding each respective addressing value to a base address.

Another aspect of the invention provides a computer-readable medium having instructions stored thereon for causing a suitably programmed information-

processing system to execute a method that includes loading a first vector register with addressing values, loading a second vector register with operand values, determining which, if any, element addresses of the first vector register have a value that duplicates a value in another element address, selectively adding certain elements of the second vector of operand values based on the element addresses the duplicated values, loading, using indirect addressing from the first vector register, elements from memory into a third vector register, adding values from the third vector register and the second vector of operand values to generate a result vector, and storing the result vector to memory using indirect addressing.

An iota instruction is described in U.S. Patent No. 6,308,250, entitled "Method and Apparatus for Processing a Set of Data Values with Plural Processing Units Mask Bits Generated by Other Processing Units," issued October 23, 2001 to Klausler, the description of which is incorporated herein by reference.

In some embodiments, a program such as the following example is used:

```
================================================================

/* kernel of the HMG tabletoy benchmark (with declarations) */

#define  LTABSIZE    22        /* logarithm of table size (27) (22
for jobmix) */
#define  NRECGEN     100000        /* records to generate on each
pass */

#define TAB_SIZE  (1 << LTABSIZE)
double table[TAB_SIZE];

typedef struct
 {
  int index;
  double value;
 }update_t;

 update_t xdata[NRECGEN];

   ...

 /* the timed loop, recs_todo (input data) = 900000000 */

   while (recs_todo)
     {
       nrec = MIN(recs_todo, NRECGEN);
       recs_todo -= nrec;

       for (idx = 0; idx < nrec; idx++)
                  table[xdata[idx].index] += xdata[idx].value;
     }

/* Please note that there is NO ivdep on this loop. */
```

```
/* In some embodiments, change the inner update loop to:

 #pragma ivdep
       for (idx = 0; idx < nrec; idx++)
          table[xdata[idx].index]
             += xpartred_add64(xdata[idx].value,xdata[idx].index);

/* in some embodiments, results were obtained by compiling with: */
/*    cc -o toy toy.c                                            */
/* and running with:                                             */
/*    aprun -n1 -p:16m toy 900000000                             */

================================================================

*   In some embodiments, the following assembly code is used for the
bolded instruction above:

*   HMG Tabletoy update:   table[xdata.index[i]] += xdata.value[i];

*   Registers computed or loaded during RHS processing of update...

        v2      [a27,2],m0              ;IX = xdata.index[*]
        v0      cidx(a11,m0)            ;IOTA
      m1    m0|m0             ;input mask
      v1    [a28,2],m0        ;Y = xdata.value[*]

*   Generate ordered msync wait,send masks
*
*   A10 = Remaining tripcount (after this pass)
*   A11 = 1
*   A22 = SSP#
*   A26 = SSP's array offset

      a24   a22^3             ;=0 iff P3
      a25   a0<a26            ;=0 iff P0 and 1st iter, else 1
      a24   a10|a24           ;=0 iff P3 and last iteration
      a21   a22-1
```

```
1           a26   a0<a24            ;=0 iff P3 and no more iters, else 1
2           a23   a22+1
3           a21   a21&3             ;restrict shift counts to be 0..3
4           a23   a23&3
5           a22   a11<<a22          ;self-mask
6           a21   a25<<a21          ;mask for SSP to wait on
7           a23   a26<<a23          ;mask for SSP to send
8           a21   a21|a22           ;wait mask
9           a22   a22|a23           ;send mask
10
11    *   Inlined "indexed partial reduction" algorithm:  Y',M1 =
12    reduce(Y, IX),M1
13    *
14    *   Y' will contain Y or sum reduced values of Y for duplicate IX
15    values;
16    *   M1 will contain an update mask where IX values are unique and
17    also where
18    *   the Y' elements that need to be added into the update (LHS)
19    vector.
20    *
21    *   Input:
22    *     v0 = IOTA vector (0,1,2,...,63)
23    *     v1 = Y vector
24    *     v2 = IX vector
25    *       m1 = Input mask
26    *     vl = #elements in v0, v1, v2
27    *
28    *   Output:
29    *     v1 = Y' vector
30    *     v2 = IX vector
31    *     m1 = Output mask of unique IX values
32
33    CNFXSZ      =     16384       ;Size of scratch conflict analysis
34    space
35
36          s4    CNFXSZ-1
37          a29   vl
```

```
1          a45   CNFXSZ*8-8
2          v5    v2&s4,m0          ;Conflict index set masked from ix
3          m4    fill(a29)
4          m3    m1&m4             ;Clear trailing mask bits beyond VL
5          a20   CNFXSZ*8
6          a45   a63-a45
7          s28   8
8          a63   a63-a20                ;Allocate private stack space
9
10         v6    v2<<s28,m0        ;(ix<<8) to make room for IOTA
11         v4    v6|v0,m0          ;(ix<<8)|IOTA
12         a27   last(m4)          ;last valid element#
13
14   cnfxloop =  *                 ;"False positive" conflict loop
15         [a45,v5] v4,m3,ord            ;Scatter (ix<<8)|IOTA (to
16   scratch array)
17         s27   x'00ff:d
18         lsync v,v
19         v6    [a45,v5],m3       ;Gather  (ix<<8)'|IOTA'
20
21         v7    +v6>>s28,m3       ;Extract ix'
22         m2    v7==v2,m3         ;M2 excludes ix's mapping to same CNFX
23
24         v9    v6&s27,m3         ;Element #s of y sums
25         m4    v9!=v0,m2         ;Conflict map
26         m3    ~m2&m3                ;Map of remaining ix values
27
28         a6    1
29         a29   pop(m4)                ;Conflict trip count (tc)
30
31         v7    cmprss(v9,m4)          ;IOTA's that conflicts map to
32         a26   pop(m3)                ;>0 if ix's mapped to same CNFX
33         m1    ~m4&m1                 ;Exclude conflicts in final M1
34
35         a1    v7,0              ;1st iota into which to sum (iota1)
36         a8    a6<a29                 ;=1 if tc > 1
37         v7,a29      a27              ;Store safe y sum index at end
```

```
1           a6    a0<a29                ;=1 if tc > 0
2           a7    a6+a8           ;=2 if tc > 1, else tc
3
4           a2    v7,a6           ;2nd iota into which to sum (iota2)
5           a3    v7,a7           ;3rd iota into which to sum (iota3)
6
7           v8    cmprss(v1,m4)         ;y values to add into y sums
8           bz    a29,noconflict        ;If no conflicts exist
9
10          a11   v8,0            ;Get 1st 3 y values (y1,y2,y3)
11          v8,a29      s0              ;Store 0 for conflict summing at
12    end
13          a12   v8,a6
14          s3    v8,a7
15
16          $REPEAT                     ;Repeat 3 update fixes per
17    iteration
18            a5    a7<a29        ;=1 if >=0 more conflicts (another
19    iter)
20            s5    v1,a1               ;Get 3 y sums (to sum conflicts
21    into)
22            a23   a2^a1               ;Determine conflict:
23    iota2==iota1
24            a5    a7+a5
25            s6    v1,a2
26            a24   a3^a1               ;Determine conflict:
27    iota3==iota1
28            a15   a5<a29        ;=1 if >=1 more conflicts
29            s7    v1,a3
30            a25   a3^a2               ;Determine conflict:
31    iota3==iota2
32            a6    a5+a15
33
34            a16   a1            ;Save iota1
35            a1    v7,a5               ;Bottom load next iter's iota1
36            a7    a6<a29        ;=1 if >=2 more conflicts
37            a17   a2            ;Save iota2
```

```
1           a2    v7,a6                ;Bottom load next iter's iota2
2           a7    a6+a7
3           a18   a3             ;Save iota3
4
5           a13   a11
6           s1    a11
7           a11   a24?a0:a11           ;y1 if iota3==iota1, else 0
8           a3    v7,a7                ;Bottom load next iter's iota3
9           a13   a23?a0:a13           ;y1 if iota2==iota1, else 0
10          s2    a12
11          a12   a25?a0:a12           ;y2 if iota3==iota2, else 0
12
13          s11   a11
14          a11   v8,a5                ;Bottom load next iter's y1
15          s13   a13
16          s12   a12
17          a12   v8,a6                ;Bottom load next iter's y2
18
19          s4,d        s3+s11         ;y3 += (iota3==iota1)? y1 : 0
20          s3    v8,a7                ;Bottom load next iter's y3
21          s2,d        s2+s13         ;y2 += (iota2==iota1)? y1 : 0
22          s4,d        s4+s12         ;y3 += (iota3==iota2)? y2 : 0
23
24          s5,d        s5+s1               ;Sum1 += y1
25          s6,d        s6+s2               ;Sum2 += y2 [+ y1]
26          s7,d        s7+s4               ;Sum3 += y3 [+ y1] [+ y2]
27          v1,a16  s5
28          v1,a17  s6
29          v1,a18  s7
30
31        $UNTIL      a15,Z
32
33    noconflict =      *              ;Branch here if no conflicts
34          bn    a26,cnfxloop         ;Repeat if more ix's mapped to
35    same CNFX
36
37          a63   a63+a20              ;Restore stack frame
```

```
*
*   End of inlined "indexed partial reduction" algorithm.
*
*   Update LHS using unique IX mask, M1, and non-allocating
gather/scatter.
*   Use ordered (ripple) msyncs if multistreamed.
*
      msync a21,v              ;Ordered msync
      v4    [a32,v2],m1,na           ;Gather TABLE[xdata.index[*]]
      v5,d  v4+v1,m1
      [a32,v2] v5,m1,ord,na          ;scatter my updated TABLE values
      msync a22,v              ;End ordered msync
```

It is understood that the above description is intended to be illustrative, and not restrictive. Many other embodiments will be apparent to those of skill in the art upon reviewing the above description. The scope of the invention should, therefore, be determined with reference to the appended claims, along with the full scope of equivalents to which such claims are entitled. In the appended claims, the terms "including" and "in which" are used as the plain-English equivalents of the respective terms "comprising" and "wherein," respectively. Moreover, the terms "first," "second," and "third," etc., are used merely as labels, and are not intended to impose numerical requirements on their objects.